Міністерство освіти і науки України
Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №5

Технології розробки програмного забезпечення

ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Виконав:
студент групи ІА-24
Красношапка Р. О.
Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Короткі теоретичні відомості 3
Хід роботи 4
Реалізація шаблону проєктування для майбутньої системи..... 5
Зображення структури шаблону 9
Посилання на репозиторій 9
Висновок..... 9

Тема: ШАБЛОНИ «ADAPTER», «BUILDER», «COMMAND», «CHAIN OF RESPONSIBILITY», «PROTOTYPE»

Короткі теоретичні відомості

Adapter (Адаптер) – це структурний шаблон, який дозволяє об'єктам з різними інтерфейсами працювати разом. Адаптер перетворює інтерфейс одного класу на інтерфейс, який очікує інший клас. Це дозволяє інтегрувати різні компоненти в систему, не змінюючи їх внутрішню реалізацію. Адаптери часто використовуються для сумісності з існуючими бібліотеками чи старими системами.

Builder (Будівельник) – це створювальний шаблон, який розділяє процес створення складних об'єктів на кілька етапів. Це дозволяє створювати різні варіанти об'єкта за допомогою однакових кроків, але з різними параметрами. Шаблон Builder корисний для побудови об'єктів, які мають багато складових, і дозволяє керувати їх створенням поетапно.

Command (Команда) – це поведінковий шаблон, який інкапсулює запит як об'єкт, що дозволяє параметризувати методи виклику, ставити їх у чергу або записувати для подальшого виконання. Шаблон Command дозволяє розділяти запити та їх виконання, а також легко реалізовувати операції скасування (undo) чи повторення (redo). Це дозволяє зручно працювати з виконанням команд у додатках.

Chain of Responsibility (Цепочка відповідальності) – це поведінковий шаблон, який дозволяє передавати запит по ланцюгу обробників. Кожен обробник може або обробити запит, або передати його наступному обробнику в ланцюгу. Це дає змогу динамічно змінювати порядок обробки запитів, а також зменшує зв'язність між клієнтом і обробниками.

Prototype (Прототип) – це створювальний шаблон, який дозволяє копіювати існуючі об'єкти замість їх створення з нуля. За допомогою шаблону Prototype об'єкт може створити нові екземпляри на основі себе, зменшуючи накладні витрати на створення складних об'єктів. Цей шаблон корисний, коли потрібно створювати багато подібних об'єктів, але зі змінами, які важко досягти через звичайне конструювання.

Хід роботи

Project Management software (proxy, chain of responsibility, abstract factory, bridge, flyweight, client-server) Програмне забезпечення для управління проектами повинно мати наступні функції: супровід завдань/вимог/проектів, списків команд, поточних завдань, планування за методологіями agile/kanban/rup (включаючи дошку завдань, ітерації тощо), мати можливість прикріплювати вкладені файли до завдань та посилатися на конкретні версії програми, зберігати виконувані файли для кожної версії.

Основні принципи та застосування Chain of Responsibility:

1. **Передача запитів між обробниками:** Шаблон дозволяє передавати запити від одного обробника до наступного, що дає можливість централізовано управляти обробкою запитів без прив'язки до конкретного обробника.
2. **Зниження зв'язності:** Завдяки використанню ланцюга обробників, клієнт не має прямого доступу до обробників, що знижує зв'язність між клієнтським кодом і конкретною реалізацією обробників.
3. **Гнучкість у додаванні нових обробників:** Ланцюг обробників легко розширюється новими елементами без зміни існуючої логіки. Це дозволяє легко змінювати або додавати нові правила обробки запитів.
4. **Обробка запитів за умовами:** Кожен обробник може перевіряти чи відповідний запит може бути оброблений саме ним. Якщо ні — передає запит наступному обробнику. Це дає змогу реалізовувати різні типи перевірок, наприклад, для обробки помилок або авторизації.
5. **Використання в обробці подій:** Шаблон часто використовується для обробки подій в GUI-додатках або в системах обробки запитів, де запит повинен пройти через кілька етапів, наприклад, валідацію, авторизацію та виконання.

Реалізація шаблону проєктування для майбутньої системи

Шаблон **Chain of Responsibility (Цепочка відповідальності)** є поведінковим шаблоном проєктування, який дозволяє передавати запит по ланцюгу обробників. Кожен обробник може або обробити запит, або передати його наступному обробнику в ланцюгу. Це дозволяє динамічно змінювати порядок обробки запитів і зменшує зв'язність між клієнтом і обробниками.

```java
package org.example.projectmanagement.handlers;

import org.example.projectmanagement.models.Project;
import org.example.projectmanagement.models.Task;
import org.springframework.stereotype.Component;

QUIRINO
@Component
public class AgileTaskStatusHandler implements TaskStatusHandler {
    3 usages
    private TaskStatusHandler nextHandler;

    2 usages  QUIRINO
    @Override
    public void setNextHandler(TaskStatusHandler nextHandler) { this.nextHandler = nextHandler; }

    4 usages  QUIRINO
    @Override
    public void handleTaskStatus(Project project, Task task, String newStatus) {
        if ("Agile".equals(project.getMethodology())) {
            if ("COMPLETED".equals(newStatus)) {
                project.setCurrentSprint(project.getCurrentSprint() + 1);
            }
        } else if (nextHandler != null) {
            nextHandler.handleTaskStatus(project, task, newStatus);
        }
    }
}
```

Рис. 1 – Код класу AgileTaskStatusHandler

```java
package org.example.projectmanagement.handlers;

import org.example.projectmanagement.models.Project;
import org.example.projectmanagement.models.Task;
import org.springframework.stereotype.Component;

 QUIRINO
@Component
public class KanbanTaskStatusHandler implements TaskStatusHandler {
    3 usages
    private TaskStatusHandler nextHandler;

    2 usages  QUIRINO
    @Override
    public void setNextHandler(TaskStatusHandler nextHandler) { this.nextHandler = nextHandler; }

    4 usages  QUIRINO
    @Override
    public void handleTaskStatus(Project project, Task task, String newStatus) {
        if ("Kanban".equals(project.getMethodology())) {
            if ("IN_PROGRESS".equals(newStatus)) {
                project.setWorkInProgressLimit(project.getWorkInProgressLimit() - 1);
            }
        } else if (nextHandler != null) {
            nextHandler.handleTaskStatus(project, task, newStatus);
        }
    }
}
```

Рис. 2 – Код класу KanbanTaskStatusHandler

```java
package org.example.projectmanagement.handlers;

import org.example.projectmanagement.models.Project;
import org.example.projectmanagement.models.Task;
import org.springframework.stereotype.Component;

 QUIRINO
@Component
public class RupTaskStatusHandler implements TaskStatusHandler {
    3 usages
    private TaskStatusHandler nextHandler;

    2 usages  QUIRINO
    @Override
    public void setNextHandler(TaskStatusHandler nextHandler) { this.nextHandler = nextHandler; }

    4 usages  QUIRINO
    @Override
    public void handleTaskStatus(Project project, Task task, String newStatus) {
        if ("RUP".equals(project.getMethodology())) {
            if ("COMPLETED".equals(newStatus)) {
                project.setPhase("Next Phase");
            }
        } else if (nextHandler != null) {
            nextHandler.handleTaskStatus(project, task, newStatus);
        }
    }
}
```

Рис. 3 – Код класу RupTaskStatusHandler

Рис. 4 – Код інтерфейсу TaskStatusHandler

Ці класи (AgileTaskStatusHandler, KanbanTaskStatusHandler, RupTaskStatusHandler) реалізують **патерн Chain of Responsibility (Цепочка відповідальності)** для обробки змін статусу завдань залежно від методології проєкту (Agile, Kanban, RUP).

1. **Ланцюг обробників**:

   Кожен клас перевіряє методологію проєкту й виконує специфічну логіку. Якщо методологія не підходить, передає запит наступному обробнику через nextHandler.

2. **Специфічна логіка**:

   - **Agile**: Збільшує номер спринта при статусі "COMPLETED".
   - **Kanban**: Зменшує ліміт WIP при статусі "IN_PROGRESS".
   - **RUP**: Переходить на наступну фазу при статусі "COMPLETED".

```java
package org.example.projectmanagement.config;

import org.example.projectmanagement.handlers.AgileTaskStatusHandler;
import org.example.projectmanagement.handlers.KanbanTaskStatusHandler;
import org.example.projectmanagement.handlers.RupTaskStatusHandler;
import org.example.projectmanagement.handlers.TaskStatusHandler;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

 QUIRINO
@Configuration
public class TaskStatusHandlerConfig {

    QUIRINO
    @Bean
    public TaskStatusHandler taskStatusHandlerChain(AgileTaskStatusHandler agileHandler, KanbanTaskStatusHandler kanbanHandler, Ru
        agileHandler.setNextHandler(kanbanHandler);
        kanbanHandler.setNextHandler(rupHandler);
        return agileHandler;
    }
}
```

Рис. 5 – Код класу TaskStatusHandlerConfig

Зображення структури шаблону

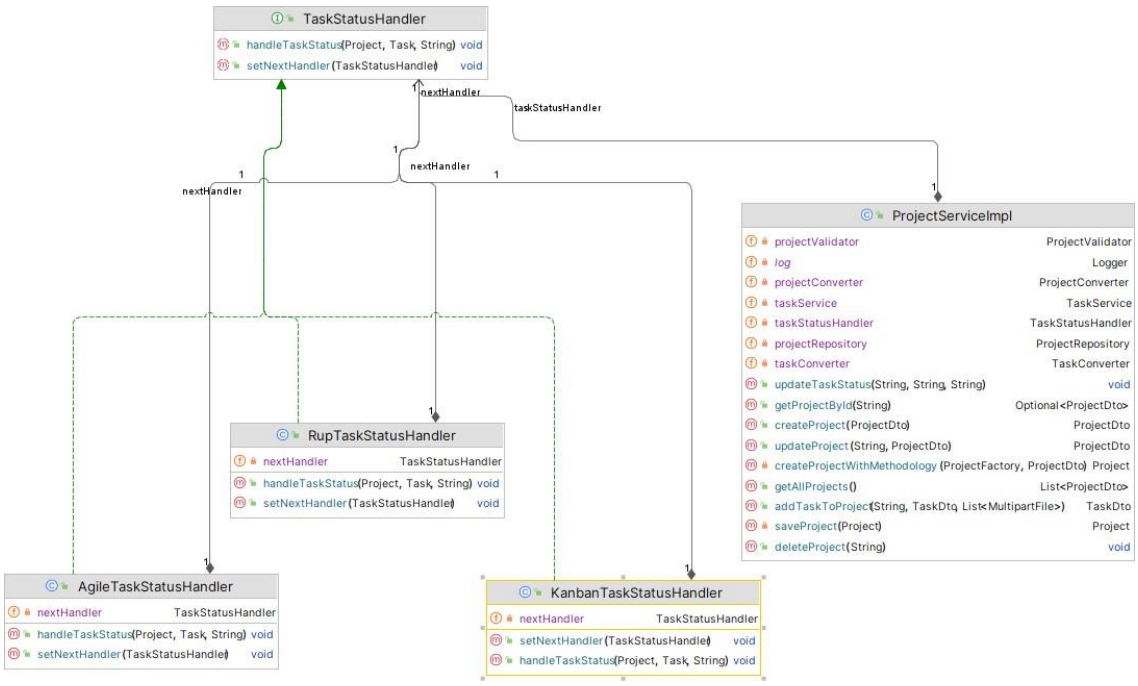

Рис. 6 – Структура шаблону

Посилання на репозиторій:

https://github.com/QUIRINO228/projectManagmentSoftware

Висновок: У результаті реалізації шаблону **Chain of Responsibility** у вигляді класів AgileTaskStatusHandler, KanbanTaskStatusHandler та RupTaskStatusHandler було продемонстровано ефективність цього патерна для обробки задач, залежно від методології проєкту.

Кожен обробник виконує специфічну для методології логіку, а у випадку, якщо він не може обробити запит, передає його наступному обробнику в ланцюзі. Це забезпечує гнучкість системи, спрощує додавання нових обробників та підтримку існуючого коду.

Використання шаблону дозволило зменшити кількість умовних конструкцій, зробивши код більш структурованим і відповідним принципам SOLID, зокрема

принципу відкритості/закритості. Отже, шаблон **Chain of Responsibility** значно покращив масштабованість і підтримуваність проєктного рішення.

.